APPENDIX B
BOOCH NOTATION

(The following materiel is extracted from Object-Oriented Analysis and Design with Application, *Second Edition, Grady Booch, The Benjamin/Cummings Publishing Company, Inc., 1994.)*

The Booch method employs a standard notation to describe a system. In the DSRS, the following elements of the Booch notation are applied: class diagram, object diagram, the relationship between classes and objects, class category diagram, and specifications.

B.1 <u>CLASS DIAGRAM</u>. A *class diagram* is used to show the existence of classes and their relationships in the logical view of a system. A single class diagram represents a view of the class structure of a system. During analysis, class diagrams are used to indicate the common roles and responsibilities of the entities that provide the system's behavior. During design, class diagrams are used to capture the structure of the classes that form the system's architecture.

B.1.1 <u>Classes</u>. Figure B-1 shows the icon used to represent a *class* in a class diagram. Its shape is that of a cloud; some call it an amorphous blob. A name is required for each class; if the name is particularly long, it can either be elided or the icon magnified. Every class name must be unique to its enclosing class category.

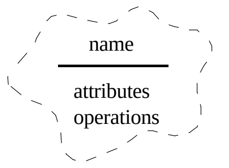

Figure B-1. Class Icon

For certain class diagram, it is useful to expose some of the *attributes* and *operations* associated with a class. At times, it is clumsy and unnecessary to show all such members. The attributes and operations associated with a class are shown after the class name and preceded by a separating line.

B.1.2 Class Relationship. Classes rarely stand alone; instead, they collaborate with other classes in a variety of ways. The essential connections among classes, summarized in Figure B-2, include *association*, *inheritance*, *"has"*, and *"using"* relationships.

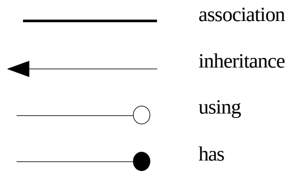

Figure B-2. Class Relationship Icon

B.1.2.1 Association. The *association* icon connects two classes and denotes a semantic connection. Associations are often labeled with noun phrases denoting the nature of the relationship. A class may have an association to itself called a *reflexive* association). It is also possible to have more than one association between the same pair of classes. Association may further adorned with their *cardinality*, as described in Table B-I.

Table B-I. Cardinality of Relationships

Cardinality Notation	Description
1	one instance only
n	unlimited number of instances
0..n	zero or more instances
1..n	one or more instances
0..1	zero or one instance
<literal>	exact number of instances
<literal>..n	exact number or more instances
<literal>..<literal>	specified range of instances

The cardinality adornment is applied to the target end of an association, and denotes the number of links between each instance of the source class and instances of the target class. Unless explicitly adorned, the cardinality of a relationship is considered unspecified.

B.1.2.2 <u>Inheritance</u>. The *inheritance* icon denotes a generalization/specialization relationship (the "is a" relationship), and appears as an association with an arrowhead. The arrowhead points to the superclass, and the opposite end of the association designates the subclass. According to the rules of the chosen implementation language, the subclass inherits the structure and behavior of its superclass. Also according to these rules, a class may have on (single inheritance) or more (multiple inheritance) superclasses; name clashes among the superclasses are also resolved according to the rules of the chosen language. In general, there may be no cycles among inheritance relationships. Also, inheritance relationship may not have cardinality adornments.

B.1.2.3 "<u>Has</u>." The *"has"* icon denotes a whole/part relationship (the "has a" relationship, also known as aggregation), and appears as an association with a filled circle at the end denoting the aggregate. The class at the other end denotes the part whose instances are contained by the aggregate object. Reflexive and cyclic aggregation is possible; aggregation does not require physical containment.

B.1.2.4 <u>Using</u>. The *"using"* icon denotes a client/supplier relationship, and appears as an association with an open circle at the end denoting the client. This relationship indicates that the client in some manner depends upon the supplier to provide certain services. It is typically used to indicate the decision that operations of the client class invoke operations of the supplier class, or have signatures whose return class or arguments are instances of the supplier class.

B.2 <u>OBJECT DIAGRAM</u>. An *object diagram* is used to show the existence of objects and their relationships in the logical design of the system. Stated another way, an object diagram represents a snapshot in time of an otherwise transitory stream of events over a certain configuration objects. Object diagrams are thus prototypical: each one represents the interactions or structural relationships that may occur among a given set of class instances, no matter what specifically named objects participate in the collaboration. In this sense, a single object diagram represents a view of the object structure of a system. During analysis, object diagrams are used to indicate the semantics of primary and secondary scenarios that provide a trace of the systems behavior. In the development process of the DSRS, object diagrams are used during analysis only. In general, object diagram is used during the design stage to illustrate the semantics of mechanisms in the logical design of the system.

B.2.1 <u>Objects</u>. Figure B-3 shows the icon used to represent an *object* in an object diagram. As is the class diagrams, a horizontal line to partition the text inside the icon into two regions, one denoting the object's name, and another providing an optional view of the object's attributes.

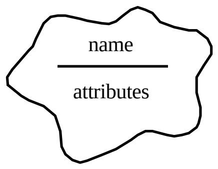

Figure B-3. Object Icon

If several object icons in the same diagram use the same object name, then they all denote the same object; otherwise, each object icon denotes a distinct object occurences. If several object icons in different diagrams use the same name, then they denote different objects, unless explicitly indicated otherwise.

B.2.2 Object Relationships. Object interacts through their links to other objects, represented by the icon in Figure B-4. A link is an instance of an association, analogous to an object being an instance of a class.

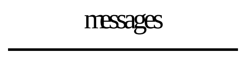

Figure B-4. Object Relationship Icon

A link may exists between two objects if and only if there is an association between their corresponding classes. This class association may manifest itself in any way, meaning that the class relationship could be a plain association, an inheritance relationship, or a "has" relationship, for example. The existence of an association between tow classes therefore denotes a path of communication (that is, a link) between instances of the classes, whereby one object may send messages to another. All classes implicitly have an association to themselves, and hence it is possible for an object to send a message to itself.

B.3 RELATIONSHIP BETWEEN CLASSES AND OBJECTS. Classes and objects are separate yet intimately related concepts. Specifically, every object is the instance of some class, and every class has zero or more instances. For practically all applications, classes are static; therefore, their existence, semantics, and relationships are fixed prior to the execution of a program. Similarly, the class of most objects is static, meaning that once an object is created, its class is fixed. In sharp contrast, however, objects are typically created and destroyed at a furious rate during the lifetime of an application.

B.4 CLASS CATEGORY DIAGRAM. Once a system grows, it is necessary to identify clusters of classes that are themselves cohesive, but are loosely coupled relative to other clusters. These clusters are directed as *class categories*. Classes and class categories may appear in the same diagram. More commonly, to represent the high-level logical architecture of the system, a class diagram that contains only class categories is used.

A class category is an aggregate containing classes and other class categories, in the same sense that a class is an aggregate containing operations and other classes. Each class in the system must live in a single class category or at the top level of the system. Unlike a class, a class category does not directly contribute state or operations to the model; it does so only indirectly, through its contained classes.

Figure B-5 shows the icon used to represent a class category. As for a class, a name is required for each class category; if the name is particularly long, it can either be elided or the icon magnified. Every class category in the model must be unique and distinct from all other class names.

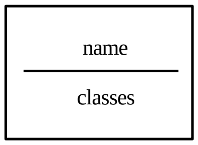

Figure B-5. Class Category Icon

For certain class category diagrams, it is useful to expose some of the classes contained in a particular class category. In most cases, however, it is unnecessary to do so.

B.5 <u>SPECIFICATIONS</u>. A *specification* is a nongraphical form used to provide the complete definition of an entity in the notation, such as a class, an association, an individual operation, or even an entire diagram. A diagram is simply a view into a model of the system under development. A specification thus serves as the nongraphical foundation model for each entity in the notation. The set of syntactic and semantic facts stated in each diagram is therefore a subset of, yet must be consistent with, the facts stated in the model's specifications.